Univerza v Ljubljani
Fakulteta za računalništvo in informatiko

Marko Novak

Uporaba globokega učenja pri pripravi hrane v malih gospodinjskih aparatih

diplomsko delo

visokošolski strokovni študijski program
prve stopnje
računalništvo in informatika

Mentor: prof. dr. Igor Kononenko
Somentor: izr. prof. dr. Matjaž Kukar

Ljubljana, 2019

Besedilo je oblikovano z urejevalnikom besedil LaTeX.

Fakulteta za računalništvo in informatiko izdaja naslednjo nalogo:

Tematika naloge:

Avtomatsko ustavljanje kuhinjskega mešalnika pri stepanju smetane je še vedno nerešen problem. Kandidat naj preveri možnost uporabe kamere za nadzor stepanja smetane v kuhinjskih mešalnikih. V ta namen naj pripravi serijo posnetkov stepanja različnih vrst smetane na konkretnem mešalniku. Posamezne fotografije naj ročno označi glede na stopnjo stepenosti smetane. Na zbranih podatkih naj preveri uspešnost detekcije stepenosti smetane z različnimi arhitekturami umetnih nevronskih mrež. Pripravi naj načrt za prototip aplikacije, ki bi omogočala avtomatsko zaustavljanje mešalnika pri želeni stopnji stepenosti smetane.

Kazalo

Seznam uporabljenih kratic

kratica	angleško	slovensko
ASIC	application-specific integrated circuit	aplikacijsko namensko integrirano vezje
API	application programming interface	vmesnik za namensko programiranje
GPIO	general-purpose input/output	splošna vhodno-izhodna enota
CNN	convolutional neural network	konvolucijska nevronska mreža
RMSE	root mean squared error	koren povprečne kvadratne napake

Povzetek

Naslov: Uporaba globokega učenja pri pripravi hrane v malih gospodinjskih aparatih

Avtor: Marko Novak

Hiter razvoj na področju vgrajenih sistemov v zadnjih letih danes omogoča vgrajevanje zmogljivih procesnih enot v mnoge naprave. Priprava hrane je eno od področij, kjer lahko uporaba takšnih tehnologij olajša vsakodnevna opravila ter prispeva k boljšim rezultatom. V diplomski nalogi smo testirali možnost uporabe kamere in umetnih nevronskih mrež za nadzor stepanja smetane v naprednih kuhinjskih mešalnikih. Najprej smo naredili več posnetkov stepanja smetane ter posamezne fotografije iz posnetkov označili glede na stopnjo stepenosti smetane. Predstavili smo nekaj pomembnejših nevronskih mrež za analizo fotografij (AlexNet, MobileNet, NasNet) ter testirali, kako se nevronske mreže, naučene na podlagi ustvarjene baze fotografij, obnesejo v praksi. Rezultati so pokazali, da lahko tako naučena nevronska mreža doseže primerno točnost, tudi ko so posnetki narejeni v drugačnih pogojih kot posnetki, ki smo jih uporabili pri učenju.

Ključne besede: umetne nevronske mreže, globoko učenje, gospodinjski aparati, analiza fotografij, smetana.

Abstract

Title: Use of deep learning in kitchen appliances to aid food preparation

Author: Marko Novak

Rapid development of embedded systems in recent years allows us to integrate powerful processing units in most electronic devices. Food preparation is one such field, where technology can ease everyday chores, and help us achieve better results. We've tried out various ways to use a camera in combination with artificial neural networks to control kitchen mixer when making whipped cream. First we made several recordings of cream during mixing. We've labelled each frame according to how well-mixed the cream is. We show some higher-importance neural networks for image analysis (AlexNet, MobileNet, NasNet) and test how well those neural networks perform after being trained on our dataset. Our results indicate that such neural networks are able to give an accurate prediction, even on photos captured under different conditions than the training data.

Keywords: artificial neural networks, deep learning, kitchen appliances, image analysis, cream.

Poglavje 1

Uvod

Pametne naprave so vedno bolj prisotne v vsakdanjem življenju. Medtem ko tehnologija razmeroma hitro prodira na poslovno področje, bodisi za namen avtomatizacije oziroma optimizacije proizvodnih linij, bodisi za hitrejše in bolj učinkovito delo s podatki in informacijami v poslovnih procesih, adaptiranje tehnoloških rešitev na področju bivanja poteka veliko počasneje. Vzrok za to gre iskati predvsem v velikosti in ceni senzorjev ter ostalih računalniških komponent, potrebnih za delovanje pametnih tehnološki rešitev, ter v dejstvu, da avtomatizacija bivalnih procesov marsikateremu potrošniku ne predstavlja neposredne finančne koristi.

Z razvojem mobilnih naprav v zadnjem desetletju je na trg prišlo tudi veliko število majhnih, cenovno ugodnih, a hkrati zelo zmogljivih naprav, kar predstavlja velik potencial za preboj pametnih naprav tudi na področje bivanja. Tako ima danes že veliko gospodinjstev robotske sesalnike, pametne hladilnike in druge naprave, ki olajšajo oziroma pohitrijo vsakdanja opravila. V diplomski nalogi se bom osredotočil na uporabo tovrstnih rešitev v aparatih za pripravo hrane. Kljub temu, da tako imenovani „kuhinjski robot“, oziroma multifunkcijski mešalnik, olajša marsikatero opravilo pri pripravi hrane, je za enkrat malo naprav, ki bi znale to opraviti samostojno.

Eden od primerov praktične uporabe tovrstnega sistema je stepanje smetane. Podjetje Bosch trenutno ponuja mešalnik OptiMUM, ki omogoča samo-

dejno ustavljanje na podlagi podatkov o obremenjenosti motorja [4], vendar je delovanje rešitve v (pre)veliki meri odvisno od vrste in količine smetane, saj to dvoje precej vpliva na obremenitev motorja med mešanjem. Nasprotno se ljudje pri stepanju smetane zanašamo predvsem na vizualne podatke, kajti za ugotavljanje drugih fizičnih karakteristik vsebine mešalnika bi morali napravo med mešanjem večkrat izključiti.

Tehnološka rešitev tega problema seveda lahko uporablja tudi druge podatke, v kolikor ima mešalnik vgrajene ustrezne senzorje: temperaturo, maso in volumen vsebine, silo, potrebno za obračanje mešalnika, trajanje mešanja in druge. Ker pa je za uporabo naštetih podatkov potreben večji poseg v napravo, sem se v okviru te diplomske naloge posvetil le reševanju problema z uporabo računalniškega vida in globokega učenja na fotografijah. Testiral sem konvolucijske nevronske mreže ter napredne globoke nevronske mreže za prepoznavanje fotografij (MobileNet, NasNet) [13, 20].

V 2. poglavju so predstavljena orodja, uporabljena za zajem in procesiranje podatkov. 3. poglavje predstavi umetne nevronske mreže in pomembnejše pristope k njihovi uporabi za analiziranje slik. V 4. poglavju je opisan potek zajema fotografij za uporabo pri učenju nevronskih mrež, njihovo označevanje in generiranje učnih podatkov s popačenjem fotografij. 5. poglavje opisuje uporabljene arhitekture nevronskih mrež, njihovo učenje ter dobljene rezultate.

Poglavje 2

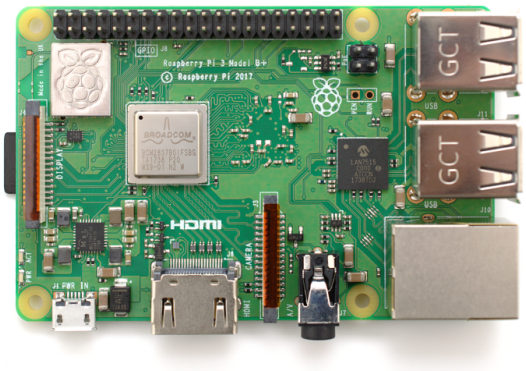
Uporabljena orodja

2.1 Zajem posnetkov stepanja smetane

2.1.1 Raspberry PI

Raspberry PI je eden od bolj popularnih računalnikov na ploščici (angl. *single-board computer*), ki v trenutno najzmogljivejši verziji 3B+ za slabih 40€ ponuja 1.4GHz ARMv8 procesor, 1GB vgrajenega spomina ter grafični procesor s podporo za strojno pospeševanje računskih operacij [19]. Za zajem posnetkov lahko uporabimo večino USB kamer, oziroma namensko kamero za Raspberry PI. Na ploščici velikosti bančne kartice (85 × 56mm, glej sliko 2.1) se nahaja tudi 40 pinov GPIO (angl. *general-purpose input/output*) za povezavo z zunanjimi napravami. Strojna oprema naprave omogoča poganjanje Linux operacijskih sistemov in s tem uporabo večine programske opreme za operacijske sisteme Linux.

Raspberry PI predstavlja osrednjo napravo v projektu Google AIY, s katerim želi podjetje širši javnosti približati uporabo umetne inteligence v vgrajenih sistemih. Podjetje trenutno ponuja komplet za prepoznavo posnetkov ter komplet za prepoznavo govora.



Slika 2.1: Raspberry PI 3. generacije [11].

2.2 Procesiranje posnetkov

2.2.1 Python

Python je visoko nivojski tolmačitveni programski jezik. Razvil ga je Guido van Rossum leta 1990, s poudarkom na berljivosti programske kode. Python je bil eden prvih programskih jezikov, ki za ločevanje ukazov in blokov kode namesto posebnih znakov uporablja presledke in prazne vrstice. Danes za razvoj jezika Python in tolmača (angl. *interpreter*) CPython skrbi Python Software Foundation, ki ju izdaja pod lastno odprtokodno licenco [12].

Python ponuja dinamične tipe in visoko stopnjo reflektivnosti (sposobnost programov, da med izvajanjem pregledujejo in spreminjajo lastno strukturo) ter omogoča programiranje z uporabo več programskih paradigem, med drugim objektno, imperativno, funkcionalno in proceduralno programiranje. Za poganjanje programov, napisanih v jeziku Python, na ciljni platformi potrebujemo le tolmača.

Python se pogosto uporablja v dinamičnih okoljih (spletni strežniki, administracija sistemov) in kot vmesnik za delo z nizko-nivojskimi knjižnicami, kjer omogoča hiter razvoj, brez potrebe po dolgotrajnem prevajanju programa ob vsaki spremembi.

Čeprav Python v začetku ni bil zasnovan za uporabo v znanstvene namene, je zaradi omenjenih lastnosti (berljivost, dinamičnost, povezljivost z nizko-nivojskimi knjižnicami) hitro zbudil interes pri matematikih in inženirjih. V ta namen se je leta 1995 med razvijalci programskega jezika Python oblikovala skupina matrix-sig, ki je skupaj z Guidom van Rossum izoblikovala namensko sintakso za delo s tabelami [16].

2.2.2 NumPy, SciPy

NumPy je knjižnica za programski jezik Python, ki vsebuje orodja za učinkovito delo s tabelami in matrikami podatkov. Jedro knjižnice je napisano v programskem jeziku C, s čimer rešuje problem hitrosti in sočasnega izvajanja Python kode pri delu z velikimi količinami podatkov.

Glavna komponenta knjižnice NumPy je NumPy matrika (angl. *NumPy array*), ki skrbi za shranjevanje in dostop do n-dimenzionalnih podatkov na pomnilniku naprave. NumPy matrika omogoča hiter dostop do poljubnih podatkov znotraj matrike, filtriranje podatkov z uporabo logičnih predikatov, matematične operacije med istoležnimi elementi v matrikah ter preoblikovanje matrike, pri čemer količina podatkov v matriki ostane enaka (npr. pretvorba iz `[x, y, 3]` matrike v matriko `[x * y, 3]`).

V primerjavi z drugimi matematičnimi orodji (prim. *Matlab, Mathematica*) NumPy ne vsebuje orodij, namenjenih analizi podatkov (matematična optimizacija, procesiranje signalov, linearna algebra). Tovrstna orodja so na voljo kot del širše zbirke SciPy [14]. Zbirka je nastala v letih 2000 - 2003, z združitvijo matematičnih orodij avtorjev Travisa Oliphanta, Erica Jonesa in Pearuja Petersona.

Velik del zbirke, vključno s knjižnico NumPy, je napisan v programskih jezikih Fortran in C in izdan pod BSD-združljivo licenco, ki dovoljuje prosto uporabo v raziskovalne in komercialne namene. Izdana je za operacijske sisteme Windows, Mac OS in sisteme z jedrom Linux, teoretično pa je združljiva z vsemi sistemi, ki podpirajo CPython in prevajalnike GCC.

2.2.3 TensorFlow

TensorFlow je orodje za strojno učenje, s podporo za delovanje v večjih porazdeljenih sistemih. Za lastno uporabo je TensorFlow razvilo podjetje Google, ki je orodje kasneje objavilo pod odprtokodno licenco. Orodje je napisano v jezikih Python, C++ in CUDA (variacija jezika C, namenjena izvajanju na grafičnih procesorjih Nvidia). Glavni API za delo z orodjem je napisan za programski jezik Python, poleg tega pa so na voljo tudi vmesniki za jezike JavaScript, C++, Java in Go [5].

TensorFlow deluje po principu programiranja s tokom podatkov (angl. *dataflow*). Osnovni element v orodju TensorFlow je usmerjen graf, v katerem vsako vozlišče vsebuje stanje (konstantno ali variabilno) ter operacijo nad vhodnimi podatki [6]. V kontekstu strojnega učenja operacije znotraj grafa predstavljajo arhitekturo modela, medtem ko stanje v grafu predstavlja *znanje* modela. Vhodni in izhodni podatki operacij znotraj grafa so predstavljeni kot tenzorji. Prednost takšnega pristopa je v tem, da večina procesnih enot na trgu danes vsebuje namenske enote za vzporedno računanje s tenzorji, kar močno pohitri izvajanje operacij. Na voljo so tudi ASIC čipi za računanje s tenzorji (angl. *tensor processing unit, TPU*).

TensorFlow graf vsebuje eno ali več vhodnih vozlišč (angl. *placeholder*), ki določajo podatkovne tipe ter dimenzije tenzorjev vhodnih podatkov. Operacija v takšnih vozliščih je definirana kot identiteta: izhodni podatki operacije so enaki vhodnim podatkom. Takšna vozlišča ustvarimo s funkcijo `tf.placeholder()`. Preostala vozlišča v TensorFlow grafu ustvarimo kot operacije, ki jim podamo eno ali več izhodiščnih vozlišč. Vozlišče, ki vrne razliko med izhodnimi podatki vozlišč `vx1` in `vx2`, ustvarimo s klicem funkcije `diff = tf.subtract(vx1, vx2)` (glej izsek 2.1).

Izvajanje operacij v TensorFlow grafu poteka znotraj seje (angl. *session*). Seja skrbi za rezervacijo virov, potrebnih za izvajanje grafa. Viri so lahko lokalni (CPU, GPU) ali zunanji (porazdeljeni sistemi). Ob inicializaciji seje TensorFlow na rezervirani strojni opremi zasede potreben pomnilniški prostor za vozlišča v grafu ter nastavi ustrezno izhodiščno stanje vozlišč.

Operacije v grafu izvedemo z uporabo funcije `session.run()`, ki kot prvi argument prejme operacije, katerih rezultati nas zanimajo, kot drugi argument pa slovar oblike { `vhodno_vozlišče: tenzor_vhodnih_podatkov` }. Pri tem lahko izpustimo vhodna vozlišča, ki niso povezana z izhodnimi operacijami (glej izsek 2.2).

Posamezne operacije oziroma veje operacij so med seboj neodvisne, zato se lahko izvajajo na ločenih napravah, kar je posebej uporabno pri učenju modelov na velikem številu podatkov. TensorFlow vsebuje tudi namenski prevajalnik, namenjen prevajanju TensorFlow grafa v strojno kodo, združljivo s programskim jezikom C++. Tako preveden graf lahko poganjamo na mnogih mobilnih in integriranih napravah.

TensorFlow je tudi eno od orodij za strojno učenje, ki podpira visoko nivojsko knjižnico Keras [7] za gradnjo in učenje nevronskih mrež.

```
import tensorflow as tf

# Nov TensorFlow graf
with tf.Graph().as_default() as graph:
    # Vhodna vozlišča
    bottlenecks = tf.placeholder(tf.float32, [None, 1056])
    truth = tf.placeholder(tf.float32, [None, 1])
    dropout_keep_rate = tf.placeholder(tf.float32)

    # Operacije
    dropout = tf.nn.dropout(bottlenecks, rate=dropout_keep_rate)
    prediction = tf.nn.dense(dropout, units=1)
    error = tf.subtract(prediction, truth)
    squared_error = tf.square(error)
    cost = tf.reduce_mean(squared_error)

    # Inicializacija nove seje za graf
    sess = tf.Session(graph=graph)

    # Inicializacija spremenljivk
    sess.run(tf.global_variables_initializer())
    sess.run(tf.tables_initializer())
```

Izsek 2.1: Inicializiranje novega TensorFlow grafa in pripadajoče seje.

```
import numpy as np

# Izvajanje operacij na grafu
def run(
    bottlenecks_input,
    truth_input=None,
    dropout_keep_rate_input=np.float32(0.5)
):
    p, c = None, None

    if truth_input:
        p, c = sess.run(
            # Končne operacije
            [prediction, cost],
            # Podatki za vhodna vozlišča
            {
                bottlenecks: bottlenecks_input,
                truth: truth_input,
                dropout_keep_rate: dropout_keep_rate_input
            }
        )
    else:
        # Vozlišče truth ni povezano z operacijo prediction,
        # zato ne potrebuje vhodnih podatkov
        p = sess.run(
            prediction,
            {
                bottlenecks: bottlenecks_input,
                dropout_keep_rate: np.float32(1)
            }
        )

    return p, c
```

Izsek 2.2: Izvajanje operacij v grafu znotraj TensorFlow seje.

Poglavje 3

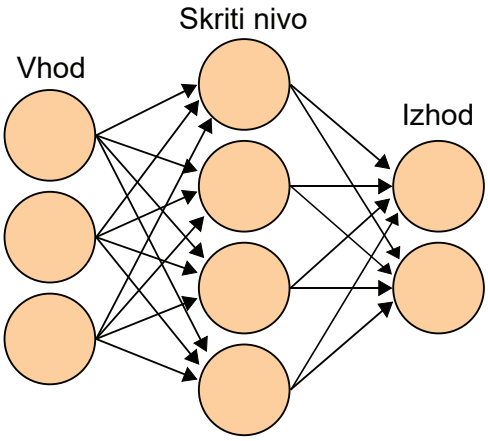
Umetne nevronske mreže

Umetne nevronske mreže predstavljajo pristop k strojnemu učenju, ki pri delovanju poskuša posnemati delovanje nevronov v živih bitjih. Navadno so predstavljene kot usmerjen graf, v katerem nevroni predstavljajo vozlišča, povezana z uteženimi povezavami. Posamezen nevron sprejme n vhodnih signalov $x_{1,\ldots,n}$, izhodni signal nevrona pa se izračuna po formuli

$$y = \theta((\sum_{i=1}^{n} x_i \, w_i) - u)$$

kjer w_i predstavlja utež povezave, θ aktivacijsko funkcijo nevrona, u pa pristranskost (angl. *bias*) nevrona.

Umetne nevronske mreže so navadno organizirane v nivoje nevronov. Število nevronov na prvem (vhodnem) nivoju nevronske mreže ustreza številu vhodnih podatkov, število nevronov na zadnjem (izhodnem) nivoju pa številu izhodnih podatkov. Med vhodnim in izhodnim nivojem imamo lahko poljubno število skritih nivojev nevronov (glej sliko 3.1). Pri učenju nevronske mreže poskušamo povezavam med nevroni določiti takšne uteži, da je odstopanje med izhodnimi signali nevronov na zadnjem nivoju in želenimi vrednostmi čim manjše.



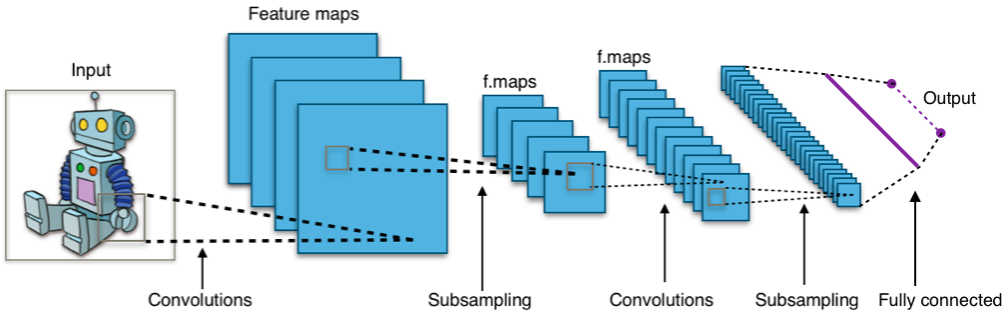
Slika 3.1: Struktura umetne nevronske mreže [10]. Nevronska mreža ima lahko več skritih nivojev.

3.1 Posebne nevronske mreže

Za reševanje specifičnih problemov s področja strojnega učenja lahko v umetnih nevronskih mrežah uporabimo posebne nevrone, ki se po izračunu izhodnega signala razlikujejo od splošnih nevronov. Pogosteje uporabljene funkcije v tovrstnih nevronih so konvolucije, združevanje (minimum, maksimum, povprečje), osip (angl. *dropout*), predimenzioniranje (angl. *reshape*) in ponavljanje (angl. *recurrency*).

3.1.1 Konvolucijske nevronske mreže

Izraz 'konvolucijske nevronske mreže' označuje umetne nevronske mreže, ki za procesiranje vhodnih signalov uporabljajo operacije konvolucije. Takšne nevronske mreže se uporabljajo predvsem za procesiranje večdimenzionalnih podatkov, najpogosteje pri analizi besedil, zvoka in fotografij. Prednost konvolucijskih nevronov je v tem, da za prepoznavo določene značilnosti kjerkoli na vhodnem signalu potrebujejo le toliko parametrov, kolikor je velikost te značilnosti. Splošni nevroni za enak rezultat potrebujejo vsaj toliko parametrov, kolikor je velikost vhodnega signala.



Slika 3.2: Struktura konvolucijske nevronske mreže [9].

Tipična konvolucijska nevronska mreža je sestavljena iz izmeničnih nivojev konvolucijskih in združevalnih nevronov, ki iz vhodnega signala izluščijo vektor značilnosti (glej sliko 3.2). Tem lahko sledi še en ali več nivojev splošnih nevronov ter izhodni nivo nevronov. Operacija konvolucije za signal x je matematično definirana kot

$$(f * g)[x] = \int_{-\infty}^{\infty} f(y)g(x-y)dy$$

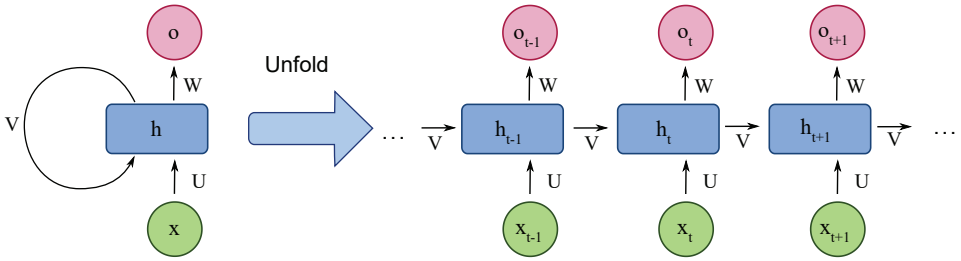
V umetnih nevronskih mrežah se pogosteje uporablja diskretna konvolucija, ki je za signal x definirana kot

$$(f * g)(x) = \sum_{y=-\infty}^{\infty} f[y]g[x-y]$$

3.1.2 Rekurenčne nevronske mreže

Rekurenčne nevronske mreže se uporabljajo pri delu z zaporedji signalov, kjer je za procesiranje signala koristno uporabiti podatke o predhodnih signalih. Takšne nevronske mreže vsebujejo rekurenčne nevrone, kar poenostavljeno pomeni, da povezave med nevroni tvorijo zanke, ki poleg signala iz predhodnega nivoja nevronov prejmejo tudi prejšnjo vrednost lastnega izhodnega signala (3.3).

Pri analizi posnetkov se pogosto uporablja kombinacija konvolucijskih in rekurenčnih nevronov. Konvolucijski nevroni služijo za prepoznavanje značil-



Slika 3.3: Levo: zanka v rekurenčni nevronski mreži; desno: razgrnjena zanka za posamezne vhodne signale [8].

nosti na posamezni fotografiji, rekurenčni nevroni pa iz zaporedja značilnosti prepoznajo širši kontekst.

3.2 Uporabljene arhitekture

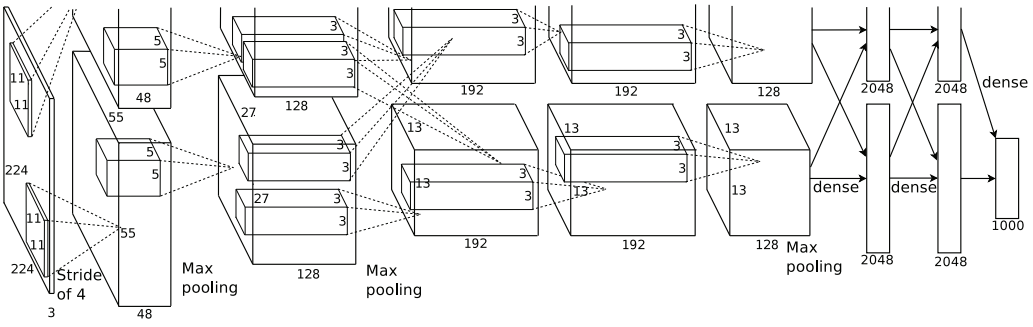
3.2.1 AlexNet

Mreža AlexNet je bila razvita z namenom uporabe konvolucije za prepoznavo in klasifikacijo fotografij višje kvalitete (originalno 224×224 točk). Arhitektura mreže je zasnovana na način, da iz vhodnih signalov ustvari dva ločena tokova ter vsak tok procesira na ločeni procesni enoti [15]. Na tekmovanju ILSVRC-2012 je mreža dosegla daleč najboljši rezultat pri klasifikaciji slik [17]. Za doseganje takšnega rezultata je mreža AlexNet uspešno uporabila sledeče koncepte:

- **Popravljena linearna aktivacija** je aktivacijska funkcija, ki aproksimira funkcijo $\theta(x) = max(0, x)$. Popravljena linearna aktivacija je določena kot $\theta(x) = ln(1 + e^x)$ in je za razliko od navadne max funkcije zvezna tudi pri vrednosti 0, kar pomeni, da jo je mogoče zvezno odvajati in integrirati. Z uporabo takšne aktivacijske funkcije so avtorji dosegli

nekajkrat hitrejše učenje ter minimizirali problem izginjajočega gradienta pri učenju nevronske mreže. Problem izginjajočega gradienta se pojavi, ker verižni odvod aktivacijskih funkcij $err'_i = (\theta'_i \circ err_{i+1}) * err'_{i+1}$, ki se uporablja za računanje napake na i-tem nivoju nevronske mreže, pri uporabi sigmoidnih aktivacijskih funkcij θ eksponentno pada, zato do učenja prihaja samo na zadnjih nekaj nivojih nevronske mreže.

- **Porazdeljeno učenje** omogoča uporabo več procesnih enot ter učinkovito uporabo sistemov z več grafičnimi karticami. Takšna zasnova je razvijalcem omogočila tudi uporabo večjega števila nevronov, saj so z uporabo več grafičnih kartic dobili tudi več razpoložljivega pomnilnika.
- **Združevanje s prekrivanjem** pomeni, da je korak pri združevanju manjši od velikosti združevalnega filtra, kar privede do prekrivanja vhodnih signalov. Takšno združevanje je manj občutljivo na točen položaj posameznih značilnosti znotraj vhodnega signala, kar pripomore k boljši prepoznavi vzorcev na vhodnih signalih.



Slika 3.4: Arhitekture mreže AlexNet [15].

Celotna arhitektura nevronske mreže vsebuje najprej 5 nivojev konvolucijskih nevronov, ki jim sledijo 3 polno-povezani nivoji nevronov. Vsak nivo je razdeljen na dva dela, z namenom, da se vsaka polovica izvaja na eni procesni enoti, zato so nevroni v konvolucijskih nivojih, z izjemo 3. konvolucijskega nivoja, povezani le z nevroni iz iste procesne enote (glej sliko 3.4).

Na vhodnih podatkih iz baze ILSVRC-2012 je nevronska mreža AlexNet dosegla napako 16.4% pri klasifikaciji fotografij za 5 najverjetnejših razredov. Klasifikacija, ki jo je nevronska mreža označila za najverjetnejšo, je bila pravilna v 65.8% primerov [17].

Nevronska mreža AlexNet, kljub svojemu pomembnemu prispevku k razvoju nevronskih mrež za analizo fotografij, zaradi velikega števila parametrov in s tem povezane računske zahtevnosti ni primerna za uporabo na mobilnih in vgrajenih napravah. Za takšno uporabo je danes na voljo precej namenskih nevronskih mrež, ki poleg hitrejšega delovanja ponujajo tudi boljše rezultate.

3.2.2 MobileNet

MobileNet predstavlja skupino nevronskih mrež, ki uporabljajo konvolucije, ločljive po globini. To so konvolucije v treh dimenzijah, za katere velja, da je njihovo konvolucijsko jedro dimenzije $[h_k, w_k, d]$ možno zapisati kot produkt dveh konvolucijskih jeder dimenzij $[h_k, w_k, 1]$ in $[1, 1, d]$.

Navadni dvodimenzionalni konvolucijski nevroni sprejmejo vhodni signal dimenzije $[h_I, w_I, d]$ ter nad njim izvedejo konvolucijo z jedrom velikosti $[h_k, w_k, d]$. Računska zahtevnost konvolucijskega nivoja, ki vsebuje K takšnih nevronov, je enaka

$$O = K * h_k * w_k * d * (h_I - h_k + 1) * (w_I - w_k + 1) \tag{3.1}$$

Konvolucje, ločljive po globini, so sestavljene iz dveh nivojev nevronov. Prvi nivo sprejme vhodni signal dimenzije $[h_I, w_I, d]$. Vsak nevron na tem nivoju sprejme eno od d plasti vhodnega signala in nad njo izvede konvolucijo z jedrom dimenzije $[h_k, w_k, 1]$, torej je računska zahtevnost tega nivoja enaka

$$O_1 = h_k * w_k * d * (h_I - h_k + 1) * (w_I - w_k + 1)$$

Drugi nivo sprejme vhodni signal dimenzije $[(h_I - h_k + 1), (w_I - w_k + 1), d]$ in nad njim izvede K konvolucij z jedrom dimenzije $[1, 1, d]$, zato je računska

zahtevnost drugega nivoja enaka

$$O_2 = K * d * (h_I - h_k + 1) * (w_I - w_k + 1)$$

Iz tega sledi, da je celotna računska zahtevnost takšne konvolucije enaka

$$\begin{aligned} O = &\ O_1 + O_2 \\ = &\ h_k * w_k * d * (h_I - h_k + 1) * (w_I - w_k + 1) + \\ &\ + K * d * (h_I - h_k + 1) * (w_I - w_k + 1) \\ = &\ (K + h_k * w_k) * d * (h_I - h_k + 1) * (w_I - w_k + 1) \end{aligned} \tag{3.2}$$

Če primerjamo računski zahtevnosti 3.1 in 3.2, vidimo, da imamo v prvem primeru faktor $K * h_k * w_k$, v drugem pa le $K + h_k * w_k$. Na ta način nevronske mreže tipa MobileNet dosegajo bistveno manjšo računsko in prostorsko zahtevnost v primerjavi z navadnimi konvolucijskimi nevronskimi mrežami.

Poleg posebnih konvolucijskih nivojev imajo nevronske mreže tipa MobileNet dva parametra, faktor ločljivosti $\rho \in (0, 1]$ in faktor globine $\alpha \in (0, 1]$. Faktor ločljivosti določa dimenzijo vhodnega signala kot $[\rho * 224, \rho * 224, 3]$, faktor globine pa za posamezni nivo konvolucije določa globino vhodnega in izhodnega signala, oziroma število konvolucijskih jeder $k_n = \alpha * K_n$, kjer K_n predstavlja število konvolucijskih jeder na n-tem nivoju v osnovni arhitekturi MobileNet [13].

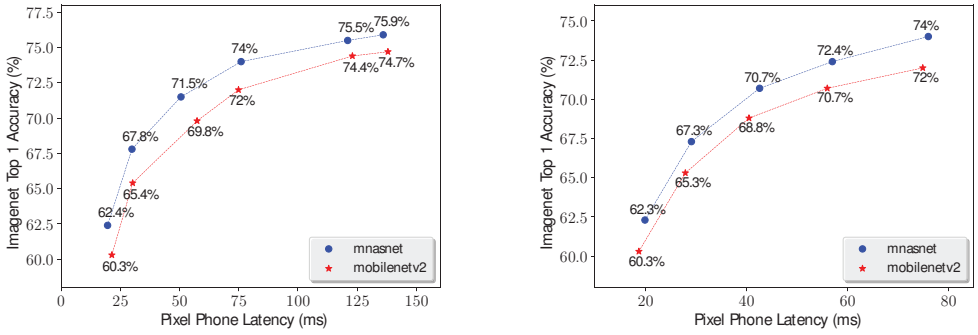
Konvolucijske mreže tipa MobileNet so posebej prilagojene uporabi na mobilnih in vgrajenih napravah, kjer lahko z uporabo parametrov ρ in α mreže prilagodimo različno zmogljivi strojni opremi in poiščemo ustrezen kompromis med točnostjo in računsko zahtevnostjo.

3.2.3 MnasNet

MnasNet nevronska mreža je konvolucijska nevronska mreža, razvita z uporabo iskanja nevronskih arhitektur (angl. *neural architecture search, NAS*). Takšen pristop pomeni, da za razvoj arhitekture nevronske mreže uporabimo algoritem, ki zna samostojno zgraditi in testirati nevronske mreže ter na podlagi ocenjevalne funkcije najti optimalno arhitekturo za dani problem. Pri

iskanju optimalne nevronske mreže so se avtorji osredotočili na dva vidika: točnost ter hitrost izvajanja na mobilnih napravah [18].

Ogrodje nevronske mreže MnasNet je sestavljeno iz zaporednih blokov, določenih s parametri $[c, k, p, f, n]$. Vsak blok vsebuje n identičnih nivojev nevronov, ki so določeni kot konvolucija tipa c (*navadna konvolucija, konvolucija, ločljiva po globini, ...*), z jedrom velikosti k in združevalno funkcijo p (*max, avg, ...*). Na koncu vsakega bloka je konvolucija s filtrom velikosti f, ki definira izhodni signal bloka.



Slika 3.5: Točnost mreže MnasNet. Levo: točnost za faktor globine 0.35, 0.5, 0.7, 1.0, 1.3, 1.4; desno: točnost za vhodno ločljivost 96, 128, 160, 192 in 224 točk [18].

Z uporabo iskanja optimalne rešitve znotraj definiranega iskalnega prostora (angl. *search space*) je avtorjem uspelo ustvariti arhitekturo nevronske mreže, ki ob primerljivi hitrosti izvajanja dosega bistveno boljšo točnost kot primerljiva nevronska mreža tipa MobileNet (glej sliko 3.5).

Poglavje 4

Zajem in označevanje podatkov

Pri strojnem učenju so ključnega pomena učni podatki. V okviru diplomske naloge en vhodni signal predstavlja fotografija vzorca na površini smetane, pripadajoč izhodni signal pa stopnja stepenosti smetane za to fotografijo.

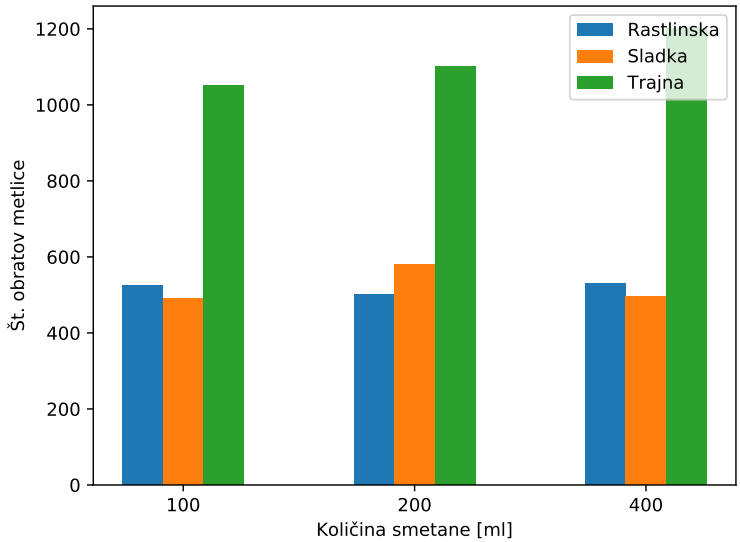
4.1 Zajem posnetkov

Pri zajemu posnetkov je pomembno, da je na slikah čim manj šuma. Stepanje smetane zato predstavlja izziv, saj se metlica za stepanje premika preko celotne posode in za seboj pušča razburkano smetano. Za kvalitetne fotografije je torej potrebno spremljati položaj metlice in fotografijo zajeti v točki, ko se metlica umakne izven vidnega polja kamere.

V ta namen sem na Raspberry PI GPIO priključil stikalo, ki ga metlica sproži vsakič, ko naredi en krog po posodi. Na podlagi signala, ki ga sproži stikalo, program za zajem posnetkov izračuna čas, potreben za vsak obhod metlice med stepanjem. Z ustreznim zamikom nato program zajame fotografijo površine smetane.

Pri zajemu ene fotografije ob vsakem obhodu metlice, program od začetka do konca stepanja smetane zajame med 500 in 1500 fotografij. Število fotografij je odvisno predvsem od vrste smetane, saj imajo različne vrste smetane različno trajanje stepanja (glej sliko 4.1). Shranjevanje takšnega števila fo-

tografij v surovi oblik zahteva veliko prostora. Pri ločljivosti 640×480 točk posamezna fotografija zasede med 100 in 300 KB prostora, torej bi za shranjevanje vseh fotografij v takšni obliki potrebovali med 50 in 450 MB prostora. Na vgrajenih sistemih je prostor za shranjevanje pogosto omejen tako glede kapacitete, kot tudi glede hitrosti. Za optimizacijo prostorske zahtevnosti program posamezne fotografije zato najprej združi v posnetke, ki jih shrani z uporabo mp4 kompresije. Zaporedne fotografije so med seboj podobne, zato posnetek celotnega stepanja smetane zasede le slabih 15 MB.



Slika 4.1: Trajanje stepanja glede na količino in vrsto smetane. Dejanski čas stepanja je odvisen od nastavljene hitrosti mešalnika.

4.2 Označevanje posnetkov

Označevanje učnih podatkov za strojno učenje je pogosto zamuden in zahteven postopek, ki lahko traja tudi več mesecev. V primeru stepanja smetane lahko ta postopek poenostavimo s sledečimi predpostavkami:

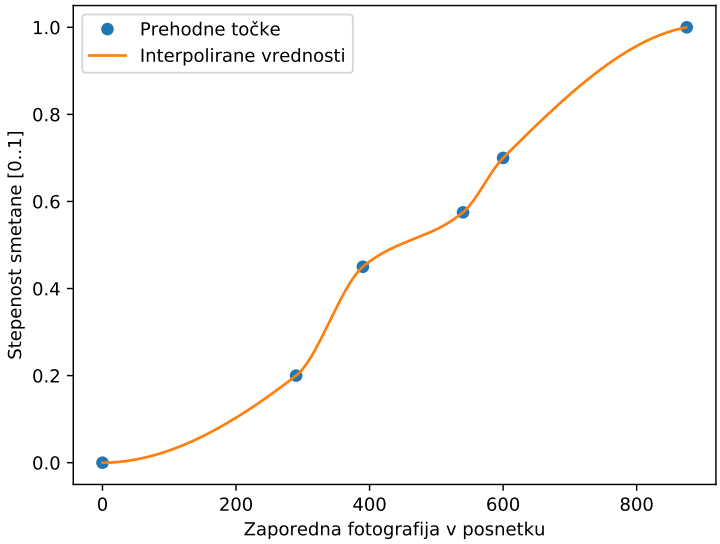
- Na fotografijah imamo samo posodo in smetano.
- Položaj posode na fotografijah iz istega posnetka se ne spreminja.
- Fotografije v posnetku si sledijo v naraščajočem zaporedju; na vsaki naslednji fotografiji je smetana bolj stepena.

Ob upoštevanju zgornjih predpostavk ugotovimo, da pri označevanju posnetkov ni potrebno posebej označevati posode, saj ji zaradi njene konstante pojave nevronska mreža ne bo pripisala posebnega pomena v procesu učenja. Prav tako ugotovimo, da lahko na posnetku določimo le ključne mejnike: točko, ko se na površini smetane začnejo pojavljati vidni vzorci, točko, ko ti vzorci postanejo bolj izraziti, točko, ko je smetana optimalno stepena, in točko, ko se na smetani začnejo pojavljati znaki prehajanja v maslo. Iz teh podatkov nato interpoliramo vrednosti za preostale fotografije v posnetku z uporabo kubičnega Hermitovega interpolatorja, ki za podane točke vrne zvezni približek linearne interpolacije (glej sliko 4.2).

V primeru, da interpolirane vrednosti za določen posnetek očitno odstopajo od vrednosti za podobne fotografije iz drugih posnetkov, to popravimo z ustreznimi dodatnimi točkami. Ob pregledu označenih fotografij odstranimo, oziroma posebej označimo nejasne fotografije ter fotografije, na katerih so vidni tuji elementi.

4.3 Generiranje dodatnih podatkov

Umetne nevronske mreže imajo veliko učljivih parametrov (uteži); nevronska mreža AlexNet, namenjena klasifikaciji fotografij, ima preko 60 milijonov parametrov [15]. Informacijski prispevek posameznega učnega primera je bistveno manjši, največ toliko bitov informacije, kot jih potrebujemo za enoličen zapis izhodnega signala. Brez dovolj velikega števila učnih primerov se zato parametri v nevronskih mrežah hitro preveč prilagodijo specifičnim vhodnim signalom. Pridobivanje dovolj velikega števila učnih podatkov je



Slika 4.2: Intepolacija vrednosti za posnetek stepanja smetane z uporabo kubičnega Hermitovega interpolatorja. Označene so točke (z leve): začetek stepanja, prvi jasni vzorci na površini, spodnja meja stepenosti, zgornja meja stepenosti, prehajanje v maslo, konec stepanja (grudice se sprijemajo v kepo masla).

časovno zahteven proces, zato se v praksi pogosto poslužujemo transformacij, s katerimi iz obstoječih vhodnih podatkov generiramo nove, enakovredne vhodne podatke.

Prva transformacija, ki jo uporabljamo, je rotacija in zrcaljenje fotografij. Vsako fotografijo lahko rotiramo med 0 in 180° ter zrcalimo preko ene ali obeh osi ter tako dobimo dodatne vhodne podatke. Uporaba tako transformiranih fotografij pri učenju nevronske mreže pomaga, da se nevronska mreža nauči prepoznati vzorce ne glede na položaj kamere in smer stepanja smetane.

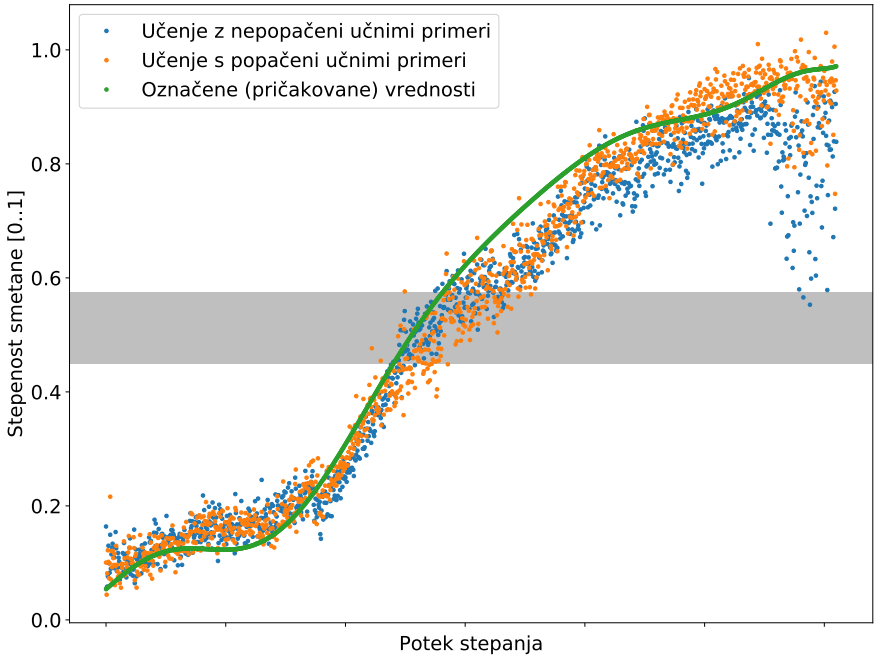
Druga transformacija je popačenje barvnih podatkov na fotografijah. Fotografije, posnete v enakih svetlobnih pogojih, imajo enak kontrast in svetlost, od nevronskih mrež pa pričakujemo, da bodo pravilno določile tudi

fotografije, posnete v drugačnih pogojih. S tem, ko posameznim učnim podatkom povečamo oziroma zmanjšamo kontrast in svetlost, se bistveno izboljša točnost modela (glej sliko 4.3).

4.3.1 Sprememba velikosti fotografij

Originalne fotografije velikosti 640×480 točk vsebujejo veliko podatkov, vendar lahko vzorce na površini smetane enako dobro vidimo tudi, če fotografije nekajkrat pomanjšamo (glej sliko 4.4). Pri učenju nevronskih mrež iz vsake originalne fotografije vzamemo naključen kvadratni izrez velikosti med 480×480 in 320×320 točk (glej sliko 4.5), ki ga pomanjšamo glede na dimenzijo vhodnega nivoja nevronske mreže. S tem iz vsake originalne fotografije dobimo 51200 različnih izsekov, na račun večje soodvisnosti med vhodnimi podatki.

Pri validaciji med učenjem nevronske mreže in pri testiranju oziroma uporabi tako naučene nevronske mreže namesto naključnih izrezov kot vhodni podatek podamo 5 izrezov velikosti 400×400 točk, skaliranih na dimenzijo vhodnega signala. Vzamemo po en izrez iz vsakega vogala fotografije ter enega iz sredine fotografije, ter kot rezultat vzamemo povprečje vseh petih izhodnih vrednosti. S tem tudi zmanjšamo verjetnost, da bi nepričakovani objekti na robovih fotografije vplivali na pravilnost rezultata nevronske mreže.



Slika 4.3: Pri drugačni postavitvi kamere je nevronska mreža, naučena samo z uporabo nepopačenih fotografij (modro) napovedala vrednosti s povprečno napako 0.078. Nevronska mreža, naučena z uporabo popačenih fotografij (oranžno), je imela za isti testni posnetek povprečno napako le 0.054. Zeleno so dejanske (označene) vrednosti za posamezno fotografijo iz testnega posnetka, sivo območje pa predstavlja optimalno stepenost smetane (glej razdelek 5.1.1).

Slika 4.4: Fotografija vzorcev na površini smetane; z leve: 100%, 50%, 25% originalne velikosti.

Slika 4.5: Izsek fotografije na sliki 4.4 velikosti 320×320 točk.

Poglavje 5

Učenje nevronskih mrež

5.1 Metodologija

5.1.1 Ocenjevanje stepenosti

Za izhodišče sem uporabil interval $[0, 1]$, tako da vrednost 0 predstavlja smetano na začetku stepanja, vrednost 1 pa skrajno točko stepanja (grudice masla se sprijemajo v kepo). Kot optimalno stepeno smetano sem določil sredino izhodiščnega intervala, vrednosti $[0.45, 0.55]$. Za označevanje fotografij sem določil še dve točki: pri vrednosti 0.2 se na smetani pojavijo prvi jasni vzorci, pri vrednosti 0.7 pa postanejo dobro vidne grudice, ki nakazujejo prehod v maslo. Pri označevanju posnetkov z uporabo takšne skale sem dosegel približno enakomerno porazdelitev fotografij čez celoten interval $[0, 1]$. Večje odstopanje od porazdelitve se je pokazalo le okrog vrednosti 0.55, saj rastlinska in trajna smetana obdržita gladko teksturo še nekaj časa po dosegu primerne stepenosti in posledično posnetek vsebuje bistveno več fotografij takšne smetane, kot posnetki navadne sladke smetane. Zaradi tega sem nekoliko spremeni skalo in premaknil zgornjo mejo za optimalno stepeno smetano na 0.575.

Končna skala, ki sem jo uporabil pri označevanju posnetkov, je sledeča (glej sliko 5.1):

- 0.000 - začetek stepanja, površina smetane je gladka in brez vzorcev
- 0.200 - na površini smetane so vidni valovi
- 0.450 - spodnja meja ustrezne stepenosti, na površini so kupčki, ki držijo obliko
- 0.575 - zgornja meja stepenosti, metlica za sabo pušča globoko sled
- 0.700 - na površini so vidne sledi grudic
- 1.000 - očitni skupki sprijetih grudic masla

5.1.2 Učni primeri

Za pripravo baze fotografij za učenje nevronskih mrež sem naredil po en posnetek stepanja 100, 200, 300, 400 in 500 mililitrov smetane za vsako od treh uporabljenih vrst smetane, skupaj 15 posnetkov stepanja. Skupno število fotografij v vseh 15 posnetkih je 11748. Fotografije sem označil, kot je opisano v razdelku 4.2 ter odstranil očitno neprepoznavne fotografije. Na ta način sem dobil bazo 11630 označenih fotografij smetane med stepanjem (glej tabelo 5.1).

Pri učenju nevronske mreže sem 90% označenih fotografij uporabil za nastavljanje parametrov nevronske mreže, preostalih 10% fotografij pa za validacijo po vsakem koraku učenja. Razporeditev fotografij v eno ali drugo skupino je psevdo-naključna, z omejitvijo, da so v obeh skupinah proporcionalno zastopane fotografije iz vseh posnetkov ter da je vsaka fotografija prisotna natanko v eni skupini.

5.1.3 Testni primeri

Za testiranjem sem naredil dodatnih 13 posnetkov, po en posnetek za 200, 300, 400 in 500 mililitrov smetane za vsako od treh vrst smetane, ter en posnetek, pri katerem sem uporabil spletno kamero s širšim vidnim kotom, tako

	Št. fotografij
Sladka smetana	
100ml	491
200ml	575
300ml	411
400ml	489
500ml	500
Rastlinska smetana	
100ml	705
200ml	807
300ml	563
400ml	637
500ml	753
Trajna smetana	
100ml	1027
200ml	1111
300ml	1111
400ml	1224
500ml	1226
Skupaj	11630

Tabela 5.1: Število fotografij v učnih posnetkih.

	Št. fotografij
Sladka smetana	
200ml	483
300ml	500
400ml	519
500ml	464
Rastlinska smetana	
200ml	626
300ml	623
400ml	665
500ml	643
Trajna smetana	
200ml	1172
300ml	1155
400ml	858
500ml	1158
Sladka smetana, druga kamera	
400ml	521
Skupaj	9387

Tabela 5.2: Število fotografij v testnih posnetkih.

da je na posnetih fotografijah viden večji del posode ter metlica mešalnika (glej sliko 5.2). Skupno sem naredil in označil 9387 fotografij za testiranje naučenih modelov (glej tabelo 5.2).

```
random_contrast = tf.image.random_contrast(
    normalized_img,
    lower=0.75, upper=1.25
)
random_brightnes = tf.image.random_brightness(
    random_contrast,
    max_delta=0.25
)
random_hue = tf.image.random_hue(
    random_brightnes,
    max_delta=0.05
)
clip = tf.clip_by_value(
    random_hue,
    0, 1
)
```

Izsek 5.1: Koda za popačenje kontrasta, svetlosti in odtenka fotografije.

5.2 Priprava učnih, validacijskih in testnih vzorcev

5.2.1 Učni vzorci

Fotografije, namenjene za učenje, najprej normaliziramo tako, da skaliramo RGB vrednosti na vrednosti $[0, 1]$. Normaliziranim fotografijam do 25% povečamo oziroma zmanjšamo kontrast in svetlost ter do 5% zamaknemo odtenek (angl. *hue*). Ker pri spreminjanju kontrasta in svetlosti fotografij lahko vrednosti za posamezne točke na fotografiji presežejo vrednosti $[0, 1]$, po opravljenih transformacijah prekoračene vrednosti omejimo (glej izsek 5.1).

Fotografije s popačenimi podatki rotiramo med $-\pi/4$ in $\pi/4$, do 50% povečamo širino in višino ter zrcalimo preko ene ali obeh osi. Na koncu

iz fotografije vzamemo izsek velikosti 480×480 točk, kar zaradi predhodne povečave ustreza velikosti izseka med 480×480 in 320×320 točk glede na izhodiščno fotografijo (glej sliko 5.3).

Za vsak krog učenja (angl. *epoch*) se iz izhodiščne fotografije generira nov učni vzorec. Na ta način se poveča raznolikost učnih vzorcev, kar preprečuje pretirano prilagajanje nevronske mreže učni množici in zagotovi, da v nobenem krogu pri učenju ne uporabimo vzorca iz iste fotografije večkrat.

5.2.2 Testni in validacijski vzorci

Generiranje testnih in validacijskih vzorcev je bistveno preprostejše. Fotografijo najprej normaliziramo na enak način, kot pri pripravi učnih vzorcev (glej razdelek 5.2.1), nato iz nje izrežemo 5 izsekov velikosti 400×400 točk, po enega iz vsakega vogala in enega iz sredine fotografije. TensorFlow za ta namen že ponuja vgrajeno funkcijo `crop_and_resize` (glej izsek 5.2).

5.2.3 Optimizacija generiranja vzorcev

Generiranje učnih vzorcev za vsak krog traja v povprečju 2 do 3 minute, pri čemer uporablja precej sistemskih virov, zato ga tudi na zelo zmogljivih sistemih težko poganjamo hkrati z učenjem nevronske mreže. Ker želimo pri učenju nevronske mreže testirati različne parametre, je smiselno učne in validacijske vzorce po generiranju shraniti na disk, da nam ob spremembi nevronske mreže ni potrebno čakati na generiranje novih vzorcev.

5.3 Ocenjevanje nevronskih mrež

5.3.1 Povprečna napaka

RMSE oziroma koren povprečne kvadratne napake pove, za koliko posamezna napovedana vrednost odstopa od pričakovane vrednosti. Izračuna se kot

$$RMSE = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(truth_i - prediction_i)^2}$$

```
off_x = 0.375   # (640 - 400) / 640
off_y = 1. / 6  # (480 - 400) / 480
boxes = np.array([
    [0, 0, 1 - off_y, 1 - off_x],   # top-left
    [0, off_x, 1 - off_y, 1],       # top-right
    [off_y, 0, 1, 1 - off_x],       # bottom-left
    [off_y, off_x, 1, 1],           # bottom-right
    [off_y / 2, off_x / 2, 1 - off_y / 2, 1 - off_x / 2],
])
patches = tf.image.crop_and_resize(
    tf.expand_dims(normalized_img, 0),
    boxes=boxes,
    box_ind=np.zeros(5),
    crop_size=[400, 400]
)
```

Izsek 5.2: Generiranje izsekov fotografije, vsak izsek je določen s pravokotnikom $[y_1, x_1, y_2, x_2]$.

kjer je n število uporabljenih vzorcev. Pri učenju nevronske mreže prilagajamo parametre nevronske mreže tako, da dosežemo čim manjši RMSE.

5.3.2 Relativna napaka

RMSE je odvisen od ranga izhodnih vrednosti. Za izhodne vrednosti iz intervala $[0, 1]$ lahko pričakujemo RMSE v bližini 0.1, za nevronsko mrežo z izhodnimi vrednostmi na intervalu $[50, 100]$ pa RMSE v bližini 5.0. Za bolj splošen prikaz se uporablja R^2, ki je neodvisen od ranga izhodnih vrednosti in se izračuna kot

$$R^2 = \frac{\sum_{i=1}^{n}(truth_i - prediction_i)^2}{\sum_{i=1}^{n}(truth_i - \overline{truth})^2}$$

R^2 predstavlja razmerje med kvadratno napako napovedanih vrednosti

$$\sum_{i=1}^{n}(truth_i - prediction_i)^2$$

ter kvadratnim odklonom pričakovanih vrednosti

$$\sum_{i=1}^{n}(truth_i - \overline{truth})^2$$

Vrednost $R^2 = 1$ pomeni, da napovedane vrednosti niso nič bolj točne, kot če bi za napovedovanje uporabili preprosto povprečje. Vrednosti bližje 0 pomenijo, da so napovedi bolj pravilne.

5.4 Arhitektura in učenje

5.4.1 Preprosta CNN arhitektura

Za osnovni model sem vzel preprosto konvolucijsko nevronsko mrežo, sestavljeno iz 5 zaporednih nivojev konvolucije ter 4 polno povezanih nivojev nevronov. Takšna arhitektura grobo posnema arhitekturo mreže AlexNet (glej razdelek 3.2.1), vendar brez prilagoditve za porazdeljeno izvajanje ter z bistveno manjšim številom učljivih parametrov (glej tabelo 5.3).

Takšna nevronska mreža ima slabih 12 milijonov učljivih parametrov, zato je na začetku že po nekaj krogih učenja prišlo do pretiranega prilagajanja vhodnim podatkom. Za rešitev tega problema sem pri učenju med konvolucijske in polno povezane nivoje dodal izpadni nivo (angl. *dropout*), ki na vsakem koraku učenja naključno izpusti polovico vhodnih signalov ter zmanjšal stopnjo učenja.

Po 40 krogih učenja (skupaj 418680 učnih vzorcev) je nevronska mreža na učnih primerih dosegla RMSE 0.177 in $R^2 = 0.398$, na validacijskih primerih pa RMSE 0.163 in $R^2 = 0.351$.

Nivo	Število filtrov	Konvolucijsko jedro	Korak
conv2d1	24	5×5	$(2, 2)$
conv2d2	36	5×5	$(2, 2)$
conv2d3	48	5×5	$(2, 2)$
conv2d4	64	5×5	$(1, 1)$
conv2d5	64	3×3	$(1, 1)$
Nivo	Vhodni signali	Izhodni signali	
dense1	23104	500	
dense2	500	100	
dense3	100	20	
prediction	20	1	

Tabela 5.3: Arhitektura uporabljene nevronske mreže.

5.4.2 Arhitektura MobileNet

Za uveljavljene arhitekture nevronskih mrež TensorFlow Hub [3] ponuja module, ki vsebujejo arhitekturo nevronske mreže ter prednaučene vrednosti parametrov. Takšni moduli kot izhodni signal vrnejo vektorje značilnosti za vhodne vzorce.

Pri gradnji nevronske mreže sem uporabil modul `mobilenet_v2_100_224` [1], ki za vsak vhodni vzorec vrne 1280 značilnosti. Temu sem dodal 4 polno povezane nivoje nevronov z enakim številom izhodnih signalov, kot v razdelku 5.4.1. Prav tako sem pred polno povezanimi nivoji dodal izpadni nivo.

Ker sem skupaj z modulom MobileNet uporabil tudi prednaučene vrednosti za zaznavanje lastnosti posnetkov, sem v prvih 30 krogih učenja spreminjal le vrednosti štirih polno povezanih nivojev nevronov na koncu nevronske mreže. Pri tem je nevronska mreža dosegla RMSE 0.074 in $R^2 = 0.071$ na učnih vzorcih in RMSE 0.077 ter $R^2 = 0.082$ na validacijskih vzorcih.

V preostalih 10 krogih učenja sem prilagajal tudi vrednosti parametrov znotraj MobileNet modula, pri čemer sem uporabil manjšo stopnjo učenja. Na koncu je nevronska mreža na učnih vzorcih dosegla RMSE 0.071 in $R^2 =$

0.068, na validacijskih vzorcih pa RMSE 0.075 in $R^2 = 0.077$.

5.4.3 Arhitektura NasNet

Učenje nevronske mreže z arhitekturo NasNet sem opravil na enak način, kot pri arhitekturi z osnovo MobileNet (glej razdelek 5.4.2), le da sem kot osnovo uporabil modul `nasnet_mobile` [2], ki za vhodne vzorce vrne 1056 značilnosti. Po prvih 30 krogih učenja (samo učenje končnih polno povezanih nivojev) je nevronska mreža dosegla RMSE 0.068 in $R^2 = 0.062$ na učnih vzorcih in RMSE 0.075 ter $R^2 = 0.077$ na validacijskih vzorcih. Po dodatnih 10 krogih učenja celotne nevronske mreže je bil RMSE na učnih vzorcih 0.067 in $R^2 = 0.060$, na validacijskih vzorcih pa 0.073 in $R^2 = 0.073$.

5.5 Rezultati učenja

V vsakem učnem krogu je bilo 10467 učnih vzorcev (po eden za vsako od 90% označenih fotografij), razdeljenih v serije po 50. Za validacijo ob učenju je bilo uporabljenih 5815 ločenih vzorcev (10% označenih fotografij, 5 vzorcev na fotografijo). Na sliki 5.4 je prikazana napaka med učenjem za učne in validacijske vzorce. Iz slike je dobro razvidna optimizacija, ki so jo posamezne nevronske mreže dosegle pri učenju.

Učenje nevronskih mrež sem izvajal preko Amazon Web Services z uporabo procesne enote Nvidia Tesla V100 s 16 GB pomnilnika. Testiranje sem izvajal lokalno z uporabo procesne enote Nvidia GeForce GTX 1060 s 3 GB pomnilnika. Časi izvajanja nevronskih mrež pri testiranju (glej tabelo 5.4) kažejo, da lahko uporabljene nevronske mreže dosežejo hitrost, potrebno za uporabo v realnem času. Za uporabljen mešalnik je zgornja meja (čas, ki je na voljo med obrati metlice) 500ms na fotografijo, oziroma 100ms za posamezni vzorec.

	Učenje	Testiranje
Preprost CNN	1775s (4.24ms / vzorec)	89s (1.90ms / vzorec)
MobileNet	2221s (5.30ms / vzorec)	179s (3.81ms / vzorec)
NasNet	2966s (7.08ms / vzorec)	288s (6.14ms / vzorec)

Tabela 5.4: Časi učenja (z uporabo Tesla V100, 40×10467 vzorcev) in testiranja (z uporabo GTX 1060, 9387×5 vzorcev).

5.6 Testiranje in uporaba

Nevronsko mrežo sem testiral s 4 posnetki za vsako vrsto smetane, ter z enim posnetkom, kjer sem uporabil drugačno kamero s širšim vidnim kotom (glej razdelek 5.1.3). Povprečne in relativne napaka za posamezne nevronske mreže so prikazane v tabeli 5.5. Iz rezultatov je razvidno, da sta mreži MobileNet in NasNet dosegli razmeroma majhno napako tudi pri posnetku z drugačno kamero, čeprav med učnimi podatki ni bilo posnetkov, narejenih pod enakimi pogoji.

Pri uporabi nevronske mreže se ne smemo zanašati na posamezne napovedi, saj lahko posamezna napoved zaradi napake odstopa od sosednjih napovedi. Zato aplikacija izračuna povprečje in standardni odklon za zadnjih 7 napovedanih vrednosti, kar na uporabljenem mešalniku predstavlja približno zadnjih 5 sekund stepanja. Akcija glede na stepenost smetane se sproži samo, kadar so napovedane vrednosti konsistentne, torej kadar je standardni odklon zadnjih 7 vrednosti majhen (pod 0.04). Podobno bi lahko uporabili tudi mediano zadnjih napovedi, vendar pri enakomerno razpršenih vrednostih mediana nekoliko bolj niha (glej sliko 5.5). Ker bi bil za dejansko ustavljanje mešalnika potreben poseg v samo napravo, aplikacija le izpiše sporočilo, ko je smetana primerno stepena, torej ko povprečna vrednost zadnjih 7 napovedi doseže ali preseže 0.45. Če s stepanjem nadaljujemo, aplikacija izpiše še eno opozorilo v točki, ko je stepanje nujno prekiniti, da preprečimo prehajanje smetane v maslo. To se zgodi, ko povprečna vrednost zadnjih 7 napovedi preseže vrednost 0.575.

	$RMSE$	R^2
Preprost CNN		
Rastlinska smetana	0.163	1.103
Sladka smetana	0.194	0.449
Trajna smetana	0.226	0.544
Drugačna kamera	0.710	8.131
MobileNet		
Rastlinska smetana	0.071	0.207
Sladka smetana	0.099	0.117
Trajna smetana	0.071	0.053
Drugačna kamera	0.160	0.411
NasNet		
Rastlinska smetana	0.085	0.299
Sladka smetana	0.109	0.142
Trajna smetana	0.068	0.050
Drugačna kamera	0.160	0.412

Tabela 5.5: Povprečna in relativna napaka pri testiranju naučenih nevronskih mrež.

5.6.1 Preprosta CNN arhitektura

Nevronska mreža je pri testiranju dosegla RMSE ~ 0.2 za enako postavitev kamere. Vendar pa so vrednosti R^2 večje od 1, torej je rezultat slabši, kot če bi preprosto ugibali in uporabili povprečno vrednost. Takšna nevronska mreža zato ni primerna za uporabo.

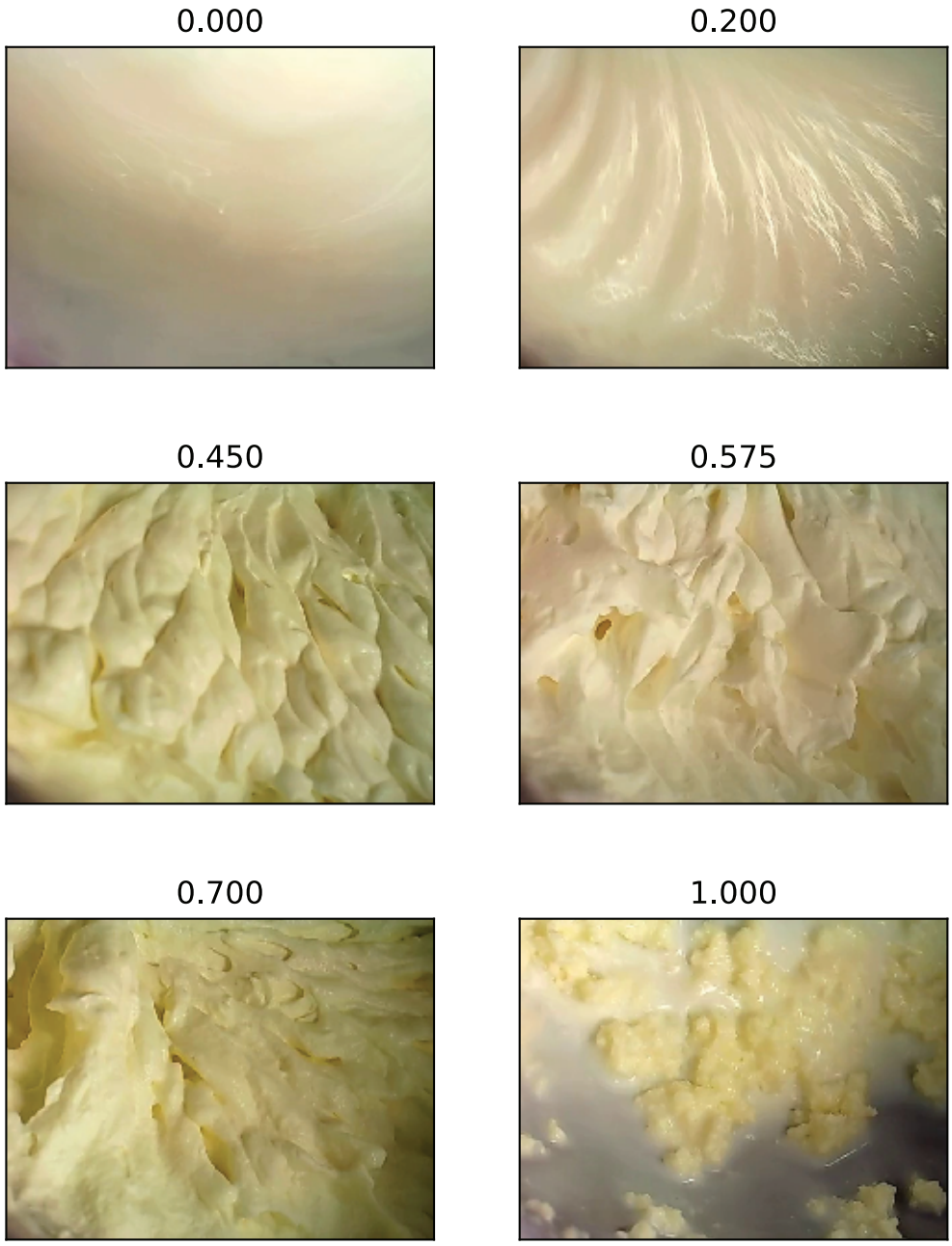
5.6.2 MobileNet in NasNet

Obe preostali nevronski mreži sta dosegli zelo podobne rezultate, tako pri učenju (glej sliko 5.4), kot pri testiranju na neodvisnih posnetkih stepanja smetane (glej tabelo 5.5). To ne preseneča, saj sta arhitekturi obeh nevronskih mrež dobro prilagojeni zaznavanju vzorcev na fotografijah. Izbira nevronske mreže za uporabo v produkcijskih rešitvah je torej odvisna predvsem od velikosti in hitrosti delovanja, za kar bi bilo potrebno obe nevronski mreži prevesti v izvedljivo kodo ter testirati na ciljni arhitekturi, saj procesne enote pogosto vsebujejo posebne prilagoditve za pospešeno izvajanje nekaterih operacij.

Zelo podobno sta se nevronski mreži obnesli tudi pri testiranju uporabe za nadzor mešalnika. Na sliki 5.6 vidimo zadnjih 7 fotografij smetane, preden je aplikacija z mrežo MobileNet signalizirala, da je smetana dosegla primerno stepenost. Slika 5.7 pa prikazuje zadnjih 7 fotografij, ko je aplikacija signalizirala, da bo smetana ob nadaljnjem stepanju začela prehajati v maslo. Zelo podoben rezultat je dala aplikacija tudi pri uporabi nevronske mreže NasNet.

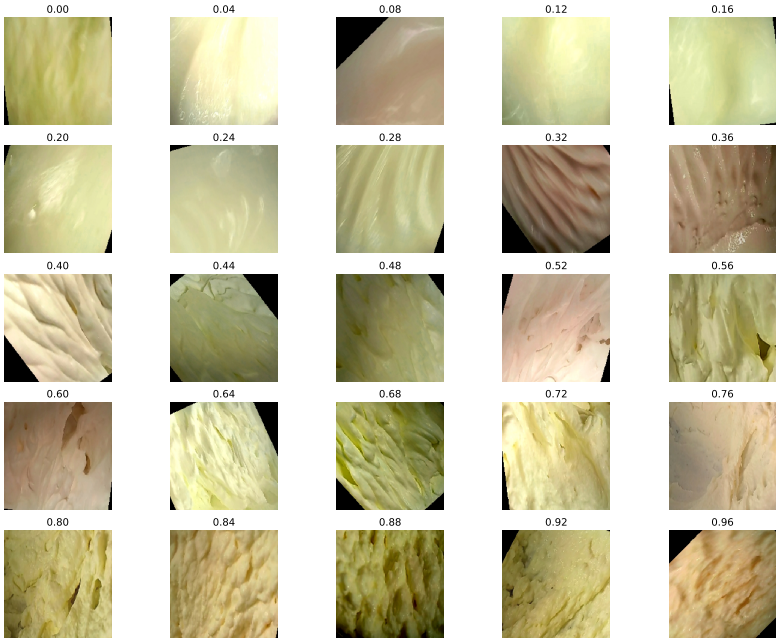
5.6.3 Napačne napovedi

Na sliki 5.8 vidimo, da so pričakovane in napovedane vrednosti pri testiranju dokaj blizu linije $y = x$, z izjemo nekaj osamelih vrednosti, kjer je napoved zelo zgrešena. Če pogledamo sliko 5.9, ki prikazuje fotografije, kjer je nevronska mreža dala napačno napoved, vidimo, da je razlog za napačno napoved v prvi vrsti nejasnost fotografij.

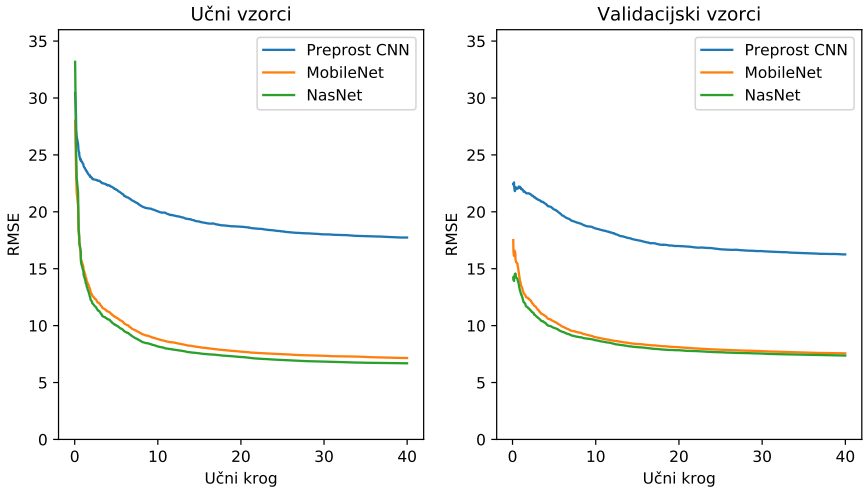


Slika 5.1: Primeri fotografij za posamezne točke na skali.

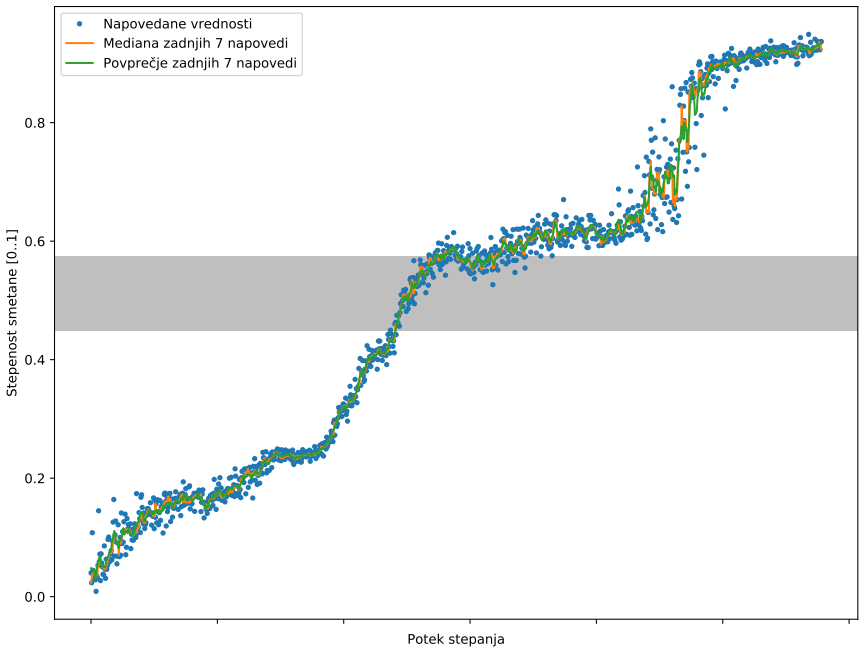
Slika 5.2: Levo: vidno polje kamere pri zajemu učnih posnetkov; desno: vidno polje spletne kamere.



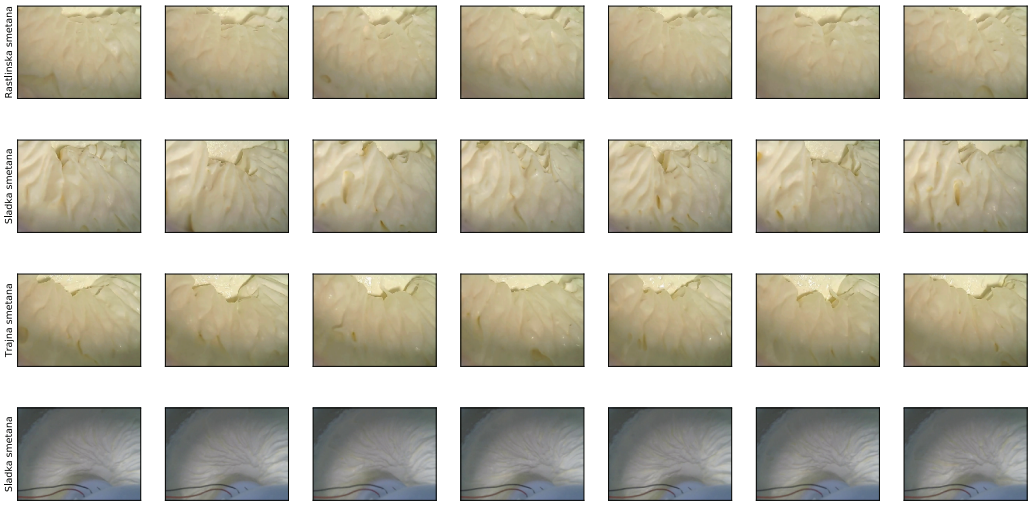
Slika 5.3: Popačeni učni vzorci.



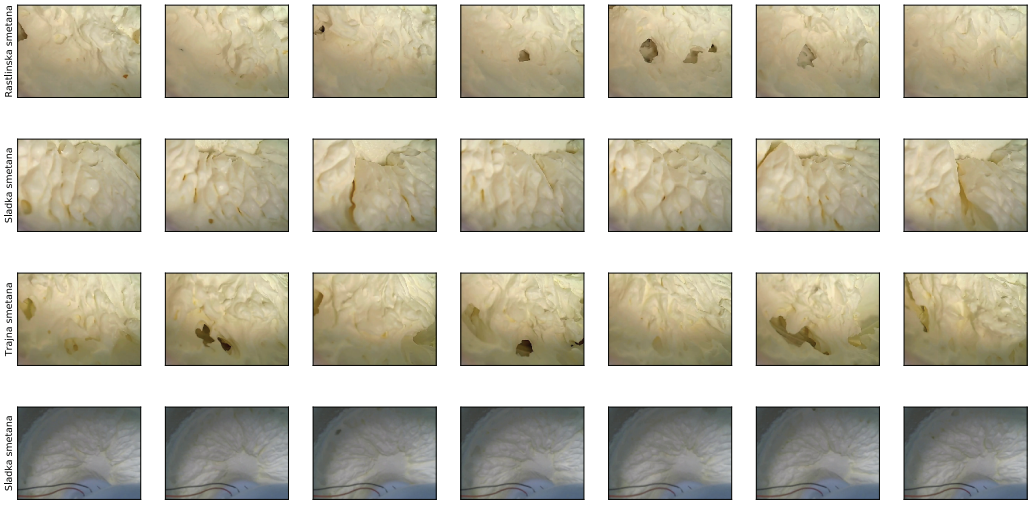
Slika 5.4: RMSE pri učenju nevronskih mrež.



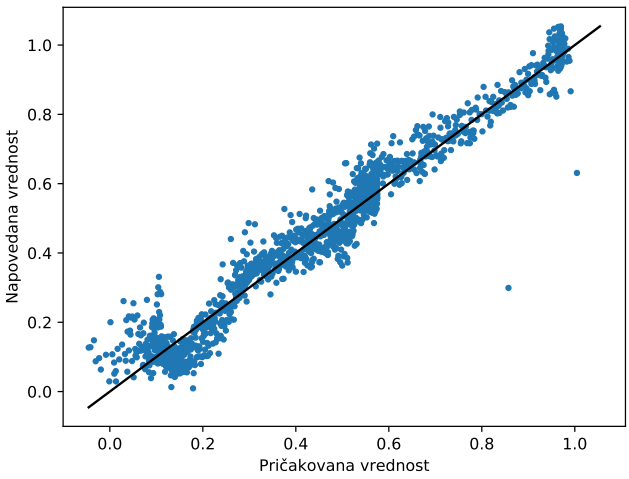
Slika 5.5: Mediana in povprečje zadnjih 7 napovedi.



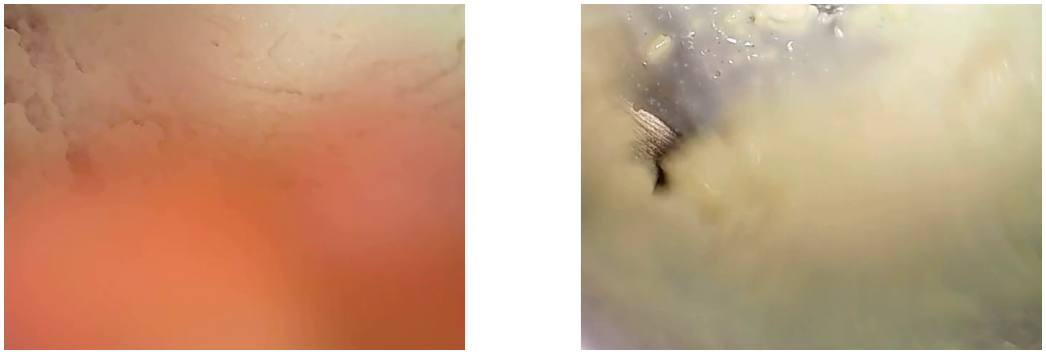
Slika 5.6: Zadnjih 7 fotografij smetane, preden je aplikacija z mrežo MobileNet ugotovila ustrezno stepenost smetane.



Slika 5.7: Zadnjih 7 fotografij smetane, preden je aplikacija z mrežo MobileNet ugotovila, da bo smetana ob nadaljnjem stepanju začela prehajati v maslo.



Slika 5.8: Graf $(truth_i, prediction_i)$ za nevronsko mrežo MobileNet.



Slika 5.9: Vhodne fotografije za osamele napačne vrednosti na sliki 5.8.

Poglavje 6

Sklepne ugotovitve

V okviru diplomskega dela je bila pripravljena baza označenih posnetkov stepanja smetane ter tri nevronske mreže, naučene z uporabo te baze fotografij. Posredno je bila v okviru diplomske naloge pripravljena tudi aplikacija, ki skrbi za zajem fotografij med mešanjem smetane ter aplikacija, ki tako zajete fotografije procesira z uporabo nevronske mreže in izpiše napovedano stopnjo stepenosti smetane za vhodne fotografije.

Obe aplikaciji sta napisani v Pythonu in pri delovanju uporabljata neoptimizirane nevronske mreže. To je dobro za namene testiranja in razvoja, kar je bil tudi cilj te diplomske naloge. Za vgradnjo v mešalnike bi bilo pametno naučene nevronske mreže prevesti v izvedljivo kodo z uporabo orodja `tfcompile` ter pripadajočo kodo za zajem in procesiranje fotografij napisati v jeziku C++. Prav tako bi bilo potrebno za uporabo na vgrajenih sistemih najti kompromis med točnostjo in hitrostjo nevronskih mrež, saj lahko manjšo točnost nevronske mreže uravnotežimo s tem, da v istem času naredimo in procesiramo večje število fotografij.

V okviru diplomske naloge sem posnetke stepanja naredil v zelo podobnih pogojih, zato sem pri učenju posnetke umetno popačil. Čeprav je na koncu naučena nevronska mreža dovolj dobro napovedala vrednosti tudi pri posnetku pod drugačnimi pogoji, bi bilo smiselno v primeru nadaljnjega razvoja tovrstne rešitve pripraviti tudi širšo bazo posnetkov pod drugačnimi

pogoji, in jih uporabiti pri učenju.

Narejena aplikacija signalizira prenehanje mešanja, ko smetana doseže spodnjo mejo stepenosti. To pomeni, da je, odvisno od vrste smetane, na voljo še nekaj časa preden smetana doseže maksimalno stepenost. Ta časovni interval je odvisen od vrste smetane, vendar na podlagi rezultatov predvidevam, da bi se dalo pripraviti rekurenčno nevronsko mrežo, ki bi na podlagi preteklih napovedi aproksimirala preostali čas stepanja smetane. Takšno napoved bi lahko kasneje uporabili za nastavljanje stopnje stepenosti smetane, oziroma zamika pri ustavljanju naprave.

Pri zajemu fotografij se je pokazala tudi ena od ovir za uporabo takšnega sistema v praksi: kapljice smetane, ki nastanejo med mešanjem, lahko pristanejo na leči kamere in s tem onemogočijo pravilno delovanje sistema. Pri kameri z majhno lečo lahko že ena kapljica popolnoma zakrije pogled. Možna rešitev za ta problem je prilagoditev nevronske mreže, in sicer tako, da poleg podatka o stepenosti smetane vrne tudi podatek, ali vhodna fotografija kaže znake, da je leča kamere umazana.

Literatura

[1] imagenet/mobilenet_v2_100_224/feature_vector, 2018. Dosegljivo: `https://tfhub.dev/google/imagenet/mobilenet_v2_100_224/feature_vector/2`, [Dostopano 25.12.2018].

[2] imagenet/nasnet_mobile/feature_vector, 2018. Dosegljivo: `https://tfhub.dev/google/imagenet/nasnet_mobile/feature_vector/1`, [Dostopano 25.12.2018].

[3] Tensorflow hub, 2018. Dosegljivo: `https://www.tensorflow.org/hub`, [Dostopano 24.12.2018].

[4] Univerzalni kuhinjski aparat optimum – avtomatsko do popolnih rezultatov, 2018. Dosegljivo: `https://www.bosch-home.com/si/novosti-in-promocije/univerzalni-kuhinjski-aparat-optimum`, [Dostopano 5.2.2019].

[5] Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S. Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Ian Goodfellow, Andrew Harp, Geoffrey Irving, Michael Isard, Yangqing Jia, Rafal Jozefowicz, Lukasz Kaiser, Manjunath Kudlur, Josh Levenberg, Dandelion Mané, Rajat Monga, Sherry Moore, Derek Murray, Chris Olah, Mike Schuster, Jonathon Shlens, Benoit Steiner, Ilya Sutskever, Kunal Talwar, Paul Tucker, Vincent Vanhoucke, Vijay Vasudevan, Fernanda Viégas, Oriol Vinyals, Pete Warden, Martin Wattenberg, Martin Wicke, Yuan Yu, and Xiaoqi-

ang Zheng. TensorFlow: Large-scale machine learning on heterogeneous systems, 2015. Dosegljivo: https://www.tensorflow.org/.

[6] Martín Abadi, Paul Barham, Jianmin Chen, Zhifeng Chen, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Geoffrey Irving, Michael Isard, Manjunath Kudlur, Josh Levenberg, Rajat Monga, Sherry Moore, Derek Gordon Murray, Benoit Steiner, Paul A. Tucker, Vijay Vasudevan, Pete Warden, Martin Wicke, Yuan Yu, and Xiaoqiang Zhang. Tensorflow: A system for large-scale machine learning. volume abs/1605.08695, 2016. arXiv:1605.08695.

[7] François Chollet et al. Keras. https://keras.io, 2015.

[8] Wikimedia Commons. File:recurrent neural network unfold.svg — wikimedia commons, the free media repository, 2017. Dosegljivo: https://commons.wikimedia.org/w/index.php?title=File:Recurrent_neural_network_unfold.svg&oldid=248564407, [Dostopano 21.12.2018].

[9] Wikimedia Commons. File:typical cnn.png — wikimedia commons, the free media repository, 2017. Dosegljivo: https://commons.wikimedia.org/w/index.php?title=File:Typical_cnn.png&oldid=266002912, [Dostopano 21.12.2018].

[10] Wikimedia Commons. File:artificial neural network.svg — wikimedia commons, the free media repository, 2018. Dosegljivo: https://commons.wikimedia.org/w/index.php?title=File:Artificial_neural_network.svg&oldid=279227349, [Dostopano 21.12.2018].

[11] Wikimedia Commons. File:raspberry pi 3 b+ (39906369025).png — wikimedia commons, the free media repository, 2018. Dosegljivo: https://commons.wikimedia.org/w/index.php?title=File:Raspberry_Pi_3_B%2B_(39906369025).png&oldid=292345672, [Dostopano 20.1.2019].

[12] Python Software Foundation. Python: History and license. Dosegljivo: https://docs.python.org/3/license.html, [Dostopano 20. 12. 2018].

[13] Andrew G. Howard, Menglong Zhu, Bo Chen, Dmitry Kalenichenko, Weijun Wang, Tobias Weyand, Marco Andreetto, and Hartwig Adam. Mobilenets: Efficient convolutional neural networks for mobile vision applications, 2017. arXiv:1704.04861.

[14] Eric Jones, Travis Oliphant, Pearu Peterson, et al. SciPy: Open source scientific tools for Python, 2001–. Dosegljivo: http://www.scipy.org/, [Dostopano 20. 12. 2018].

[15] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In F. Pereira, C. J. C. Burges, L. Bottou, and K. Q. Weinberger, editors, *Advances in Neural Information Processing Systems 25*, pages 1097–1105. Curran Associates, Inc., 2012. Dosegljivo: http://papers.nips.cc/paper/4824-imagenet-classification-with-deep-convolutional-neural-networks.pdf.

[16] K Jarrod Millman and Michael Aivazis. Python for scientists and engineers. *Computing in Science Engineering*, 13(2):9–12, March 2011. doi:10.1109/MCSE.2011.36.

[17] Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael Bernstein, Alexander C. Berg, and Li Fei-Fei. ImageNet Large Scale Visual Recognition Challenge. *International Journal of Computer Vision (IJCV)*, 115(3):211–252, 2015. doi:10.1007/s11263-015-0816-y.

[18] Mingxing Tan, Bo Chen, Ruoming Pang, Vijay Vasudevan, and Quoc V. Le. Mnasnet: Platform-aware neural architecture search for mobile, 2018. arXiv:1807.11626.

[19] Eben Christopher Upton. Raspberry pi 3 model b+ on sale at $35, 2018. Dosegljivo: https://www.raspberrypi.org/blog/raspberry-pi-3-model-bplus-sale-now-35/, [Dostopano 26. 1. 2019].

[20] Barret Zoph, Vijay Vasudevan, Jonathon Shlens, and Quoc V Le. Learning transferable architectures for scalable image recognition. In *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 8697–8710, June 2018. doi:10.1109/CVPR.2018.00907.